CLOCK TREE SYNTHESIS UNDER AGGRESSIVE BUFFER INSERTION

BY

YING-YU CHEN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Associate Professor Deming Chen

ABSTRACT

In this thesis, we propose a maze-routing-based clock tree routing algorithm integrated with buffer insertion, buffer sizing, and topology generation that is able to consider general buffer insertion locations. While previous work on buffered clock tree synthesis restricts potential buffer locations on merge nodes in the clock tree topology, our proposed algorithm has more freedom and thus achieves more robust slew control. Buffer insertion along routing paths had been mostly avoided previously due to the difficulty of maintaining a low skew under such aggressive buffer insertion. We developed an accurate timing analysis engine for delay and slew estimations and a balanced routing scheme for better skew reduction during clock tree synthesis. As a result, we can perform aggressive buffer insertion and maintain accurate delay information and low skew. Buffer sizing is also guided by its performance for slew control. Experiments show that our synthesis results not only honor the slew constraints but also maintain reasonable skew.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I thank all my family and friends for their care and support. I would like to thank my adviser, Deming Chen, and my labmates for their enlightening discussions. I would especially like to thank Chen Dong and Christine Lee for their contributions to this project. Last but not least, I want to thank Scott Deeann Chen for his helpful discussions, constant support, and great encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CTS	Clock Tree Synthesis
DME	Deferred-Merge Embedding
VLSI	Very-Large-Scale Integration

CHAPTER 1

INTRODUCTION

Clock distribution networks play an essential role in synchronous VLSI chips. The quality of the clock distribution network tremendously affects the performance of the chip, as the pace of almost every data transfer is determined by the clock signal. As a consequence, clock network synthesis has always been an important research topic over the years [1–17].

Clock tree synthesis requires accurate timing analysis in order to control clock skews among different parts of the clock tree. Also, buffer insertion is an essential part in practical clock networks, since it helps reduce delay and slew. In some clock synthesis work, unbuffered clock networks are generated alone, requiring a separate buffer insertion stage using conventional or specialized buffer insertion algorithms. In other work [6,8,10–12,16], buffer insertion and clock tree routing are integrated rather than performed separately. Therefore, the delay information can be constantly updated to guide the routing and make the clock tree more balanced.

In [7,12,16], buffer insertion is performed in order to maintain a reasonable slew in the clock network. However, potential buffer insertion locations are restricted only to merge nodes in the clock tree topology. Given more general scenarios, such buffer arrangement may not be sufficient to meet a hard slew constraint. Figure 1.1 is plotted based on a series of SPICE simulations, in which we can observe that as wire length increases, wire output slew grows dramatically. Increasing driving buffer size from $20X$ to $30X$ only provides a slight improvement to the slew. In other words, increasing buffer sizes alone cannot solve the slew control problem. For clock tree design in a large chip area, buffers have to be inserted into wire segments rather than just on merge nodes only. To the best of our knowledge, [7] is the only work that inserts buffers along the routing paths rather than solely on the merge nodes. The

differences are illustrated in Figure 1.2.

The reason why buffer insertion along routing paths was not studied intensively is due to the difficulty of maintaining accurate delay information in the bottom-up clock tree synthesis process. For example, the slew at the input of the buffer cannot be predicted based on the downstream circuit. However, the input slew affects delay and slew of the downstream circuit severely. Buffer intrinsic delay is especially sensitive to input slew and varies significantly when the input slew changes. For example, for a $10X$ buffer, the intrinsic delay can vary up to 10 ps in the 45 nm technology. Without accurate delay information, a clock tree with low skew cannot be achieved.

Our proposed algorithm is designed to balance the clock tree while allowing buffers to be inserted along routing paths. Meanwhile, buffer insertion and buffer sizing are guided by slew so that the slew in the entire clock tree is controlled under a certain constraint.

The contribution of our work is summarized as follows:

- We built a library for buffers and wires with accurate delay and slew characterization, which matches SPICE simulation results closely.
- We guarantee bounded slew in the entire clock tree through buffer insertion and buffer sizing along the routing paths.
- We maintain a low skew through accurate timing analysis and balanced routing.

Part of this work is published in [18].

The rest of the thesis is organized as follows: In Chapter 2, we review the clock tree synthesis problem and some selected previous approaches. In Chapter 3, we present the delay model we use. In Chapter 4, we explain and analyze our algorithm in detail. Experimental results are presented in Chapter 5, and Chapter 6 concludes the thesis.

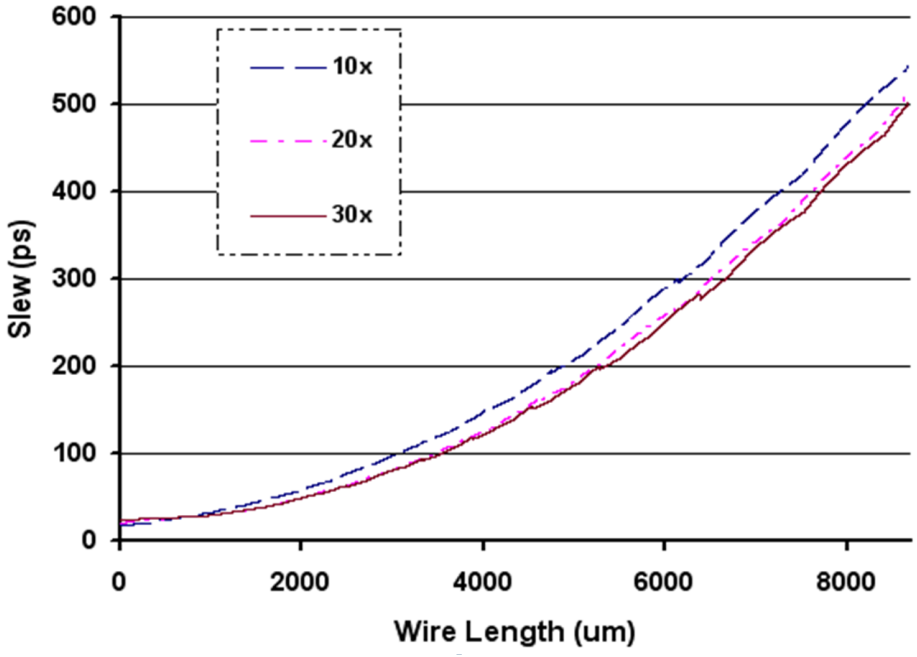

Figure 1.1: An example showing that buffer sizing is not sufficient to control slew.

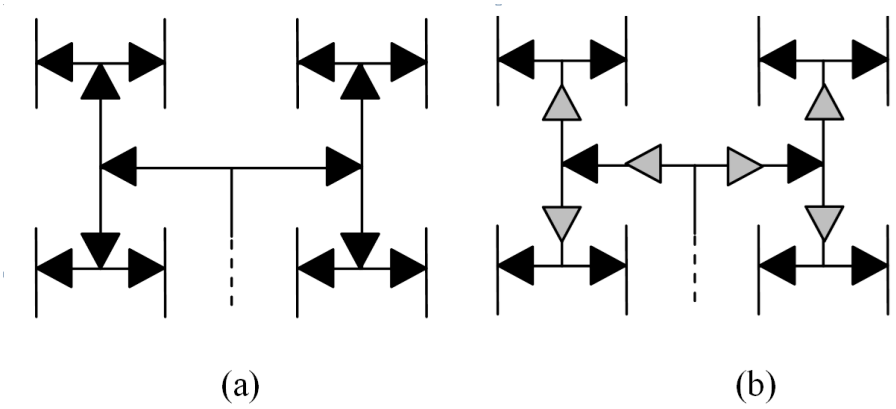

Figure 1.2: (a) A clock tree with buffer locations restricted to merge nodes. (b) A clock tree in which buffer locations have more freedom. Gray buffers are not located on merge nodes.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Preliminaries

Given a set of clock sinks, which correspond to the clock inputs of clocked elements such as flip-flops in a digital circuit, the general clock tree synthesis problem is to construct a clock distribution network in a tree topology, in which the root node represents the clock source; the leaf nodes represent the clock sinks; and the internal nodes represent the merge nodes, where subtrees are embedded. The objective may be minimizing clock skew, power dissipation, etc., varying from application to application.

Def Clock skew is defined as the maximum difference among the delays from the clock source to all of the clock sinks.

Def Latency is defined as the maximum of the delays from the clock source to all of the sinks.

Some clock tree synthesis algorithms aim to minimize the clock skew [1–3], while others bound the clock skew within a given hard constraint [4].

Usually, the clock tree is mostly binary because the algorithm of synthesizing such a tree is simpler. However, as the number of sinks may not be 2^k, a pure binary tree is not sufficient to connect all the sinks. There is also some work utilizing arbitrary number of branches at each level in order to build a symmetrical tree topology [19].

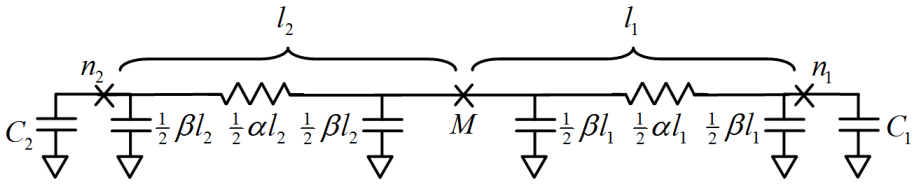

Figure 2.1: Merge segment calculation.

2.2 Fundamental CTS Algorithms

The Deferred-Merge Embedding (DME) algorithm [1] is the fundamental of many later clock tree synthesis algorithms. It consists of a bottom-up stage and a top-down stage. A tree of merge segments, which represent the possible locations of merge nodes, is constructed in the bottom-up stage. Once all the merge segments are constructed, the exact positions of merge nodes are determined in the top-down stage. Merge segments are calculated to balance the delays of the two sub-trees.

In [2], a routing algorithm is proposed to compute the optimal merge node position that results in a zero-skew clock tree based on the Elmore delay model. Because the Elmore delay model has a closed-form expression of delays, a closed-form formula can be obtained to predict the optimal merge node position. Figure 2.1 is an example of a branch of a clock tree presented in the Elmore delay model. Table 2.1 shows the notations used in merge segment calculation and necessary assumptions made to simplify the problem. We also assume that the clock skews in T_1 and T_2 are both zero. For sub-trees with non-zero skews, both an upper bound and a lower bound of delays need to be taken into consideration.

To balance the delays on the two sides, the following condition is required:

$$\alpha l_1(\frac{\beta l_1}{2} + C_1) + t_1 = \alpha l_2(\frac{\beta l_2}{2} + C_2) + t_2 \tag{2.1}$$

By establishing

$$l_1 = xl \tag{2.2}$$

$$l_2 = (1 - x)l \tag{2.3}$$

$$0 \leq x \leq 1 \tag{2.4}$$

Table 2.1: Notations in merge segment calculation.

T_1, T_2	The two sub-trees that are to be embedded in the current iteration.
n_1, n_2	The root nodes of T_1 and T_2, respectively.
M	The optimal merge node, which is to be determined.
t_1	The delay from n_1 to any of the sinks in T_1.
t_2	The delay from n_2 to any of the sinks in T_2.
C_1, C_2	The load capacitance of n_1 and n_2, respectively.
α	The unit resistance of wires.
β	The unit capacitance of wires.
l	The distance between n_1 and n_2.
l_1	The distance between n_1 and M.
l_2	The distance between n_2 and M.

we have the following condition for a zero-skew merge segment:

$$x = \frac{((t_2 - t_1) + \alpha l(C_2 + \frac{\beta l}{2})}{\alpha l(C_1 + C_2 + \beta l)} \tag{2.5}$$

The merge segment can be established by collecting all the points that have Manhattan distance of l_1 with n_1 and Manhattan distance of l_2 with n_2. The set of points should form a Manhattan arc. Such choice of merge segment is optimal when n_1 and n_2 are connected straight to the merge segment without any detours.

The zero-skew routing algorithm introduced above produces a zero-skew clock tree under the Elmore delay model. However, the Elmore delay model is a second-order approximation and is known to overestimate delays. Clock trees synthesized in this model cannot have a truly low skew in reality. To address this issue, some later clock tree synthesis algorithms modify the merge segment calculation in order to capture more complicated delay models [13–15]. Others extend the concept of merge segments to merge regions in order to establish some flexibility during the synthesis [4, 10, 11]. However, they are only valid under the assumption that no buffers are inserted and no detours are taken in the routing paths. In practical circuits, buffer insertion along the routing path can be helpful in terms of slew control.

Topology generation is a top-down process that determines the order of the sub-tree pairs that are to be embedded. While the zero-skew routing

algorithm guides the choice of the merge node position when a pair of downstream nodes are given, it does not guide the choice of the pair of nodes to be embedded. The order of embedding as well as the choice of pairs greatly affects the resulting clock tree, as illustrated in Figure 2.2.

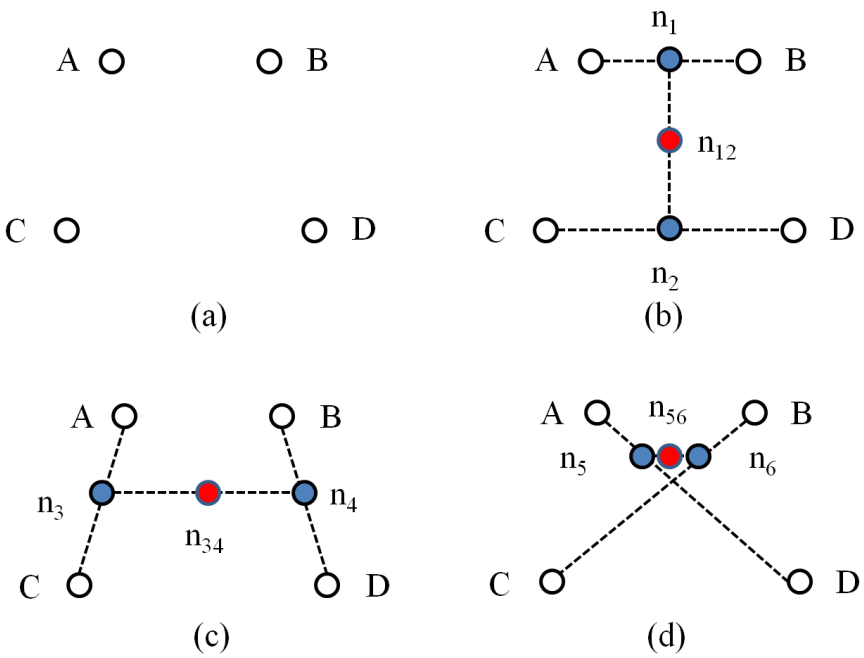

Figure 2.2: (a) The given clock sinks. (b) Merging (A, B) and (C, D), then merging (n_1, n_2). (c) Merging (A, C) and (B, D), then merging (n_3, n_4). (d) Merging (A, D) and (B, C), then merging (n_5, n_6).

The DME algorithm can be incorporated with topology generation and dynamically determine the order of the sub-tree pairs that are to be embedded, as in [3]. Initially, a nearest-neighbor graph is built to represent the feasibility of embedding each pair of sinks. When a new merge node or merge segment is created, it is added into the nearest-neighbor graph, and the nodes that are merged in the previous iteration are removed from the graph. The costs of all edges are recalculated, and thus the new node can be considered in the next nearest-neighbor selection. This procedure is repeated until there is only one node left in the nearest-neighbor graph, which is the root node of the clock tree.

CHAPTER 3

DELAY MODELING

Our goal is to match our delay and slew estimation as close to simulation results as possible while maintaining reasonable running time and space complexity. We performed a series of experiments on different delay models and on SPICE to find the characteristics of delay and slew. Based on these characteristics, we developed a delay/slew analysis scheme that is suitable for our buffered clock tree synthesis algorithm.

3.1 Insufficiency of Existing Delay Models

The Elmore delay model is widely used in clock tree synthesis to estimate interconnect delays due to its simple and closed-form expressions. It has been widely known that Elmore delay is the negative of the first moment of impulse response and can be highly inaccurate by ignoring resistive shieldings. More importantly, the Elmore delay model cannot compute the slew within an RC netlist.

Existing work [20, 21] developed closed-form delay and slew expressions of ramp inputs by matching higher order moments of the impulse response. However, approximating a real curve as a ramp signal can introduce a large amount of error. We performed an experiment intended to evaluate the difference resulting from different shapes of input waveforms. Figure 3.1 shows the circuit being measured in this experiment. Input waveforms of the same slew rate were applied at the input of the buffer B_{input}, and the output waveform was measured at B_{load}. As shown on the bottom of Figure 3.2, the input waveforms applied at B_{input} in the two measurements were a curved signal and a ramp signal, respectively. Although both of them have a 150 ps 10%-90% slew, the buffer output signal shifted by 32 ps, as shown on the

top of Figure 3.2. This study indicates that for an accurate delay estimation, the buffer input waveform needs to be considered. We also need to study the wire that drives the buffer input because different wire lengths can make the input waveform distort from a ramp input differently. In order to have accurate delay estimation for both buffers and wires, real input waveforms need to be considered.

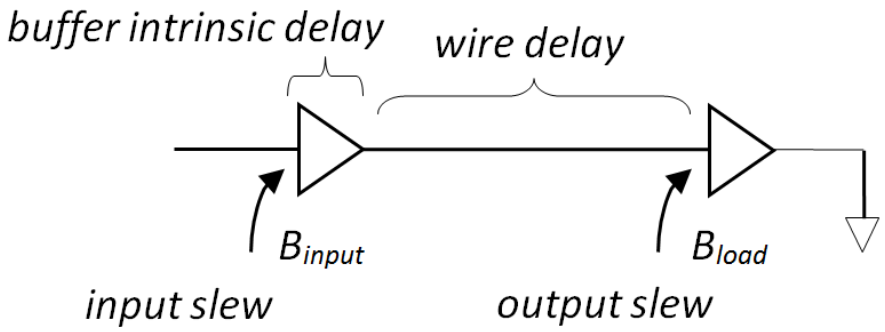

Figure 3.1: The circuit being measured in the curve vs. ramp experiment.

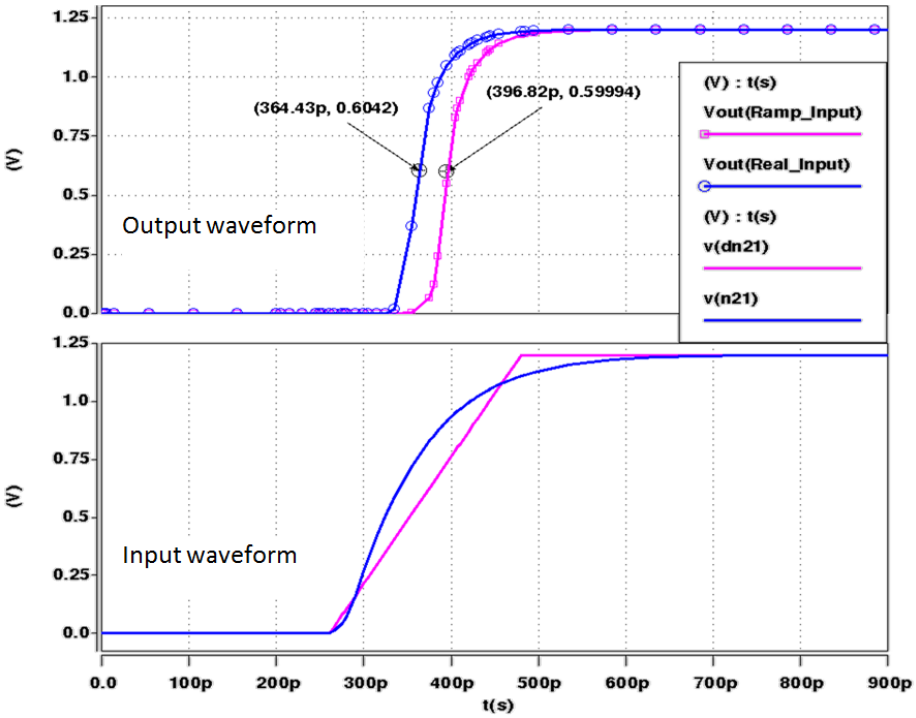

Figure 3.2: Difference between the transient responses from a curve input and a ramp input.

For comparison of accuracy, we also implemented the higher-moment-based delay models [20, 21] and compared with SPICE simulation results. Indeed, the higher-moment-based models outperform the Elmore delay model. However, because of their inability to model the effects from curved input waveform, delay estimations based on these models were still quite different from SPICE simulation results. In other words, these models could not provide sufficient accuracy for our use.

Because there is no accurate slew model to capture the wire slew propagation, our approach is to carry out SPICE simulations by varying the input slew of the driving buffer, the driving buffer type, the load wire length, and the load capacitance in order to determine the output waveform of the wire. The output waveform can be used to drive the next-stage buffer for accurate delay and slew estimation. Because SPICE simulation is time consuming, our approach is to pre-characterize the delay and slew using the aforementioned parameters and then build a multi-dimensional function as a model. The next section explains how we carry out this characterization in detail.

3.2 Delay/Slew-Library-Based Implementation

We are first given a library of buffers of different sizes. Each buffer is characterized as two cascaded inverters in a SPICE netlist. Different buffer sizes come from different widths of the transistors used in the buffers.

We partition our clock trees into smaller components with cuts on buffered nodes. The components act as units on which we perform delay and slew estimations. The circuitry structures of the components can be categorized in two types: single wire and branch.

3.2.1 Modeling Single-Wire-Type Components

All single-wire-type components start with a driving buffer. Most of them end with a load buffer and a few others end with a sink. To simplify the delay/slew estimation functions, we developed a set of functions for each combination of driving buffer type and load buffer type. Components ending

with a sink can be approximated by a component ending with a buffer of similar load capacitance. In this way, buffer type and load capacitance are considered.

We built a small circuit as depicted in Figure 3.3(a). B_{drive} and B_{load} are the driving and load buffers, which can be of different types in the buffer library. L is the wire length. We added an extra input buffer B_{input} and a wire segment of length L_{input} in order to create the special buffer output waveform as the actual input to the buffers and wires we want to measure. The input waveform is an ideal ramp, and B_{input} transforms it into the special buffer output waveform. L_{input} can be adjusted to generate a range of input slew.

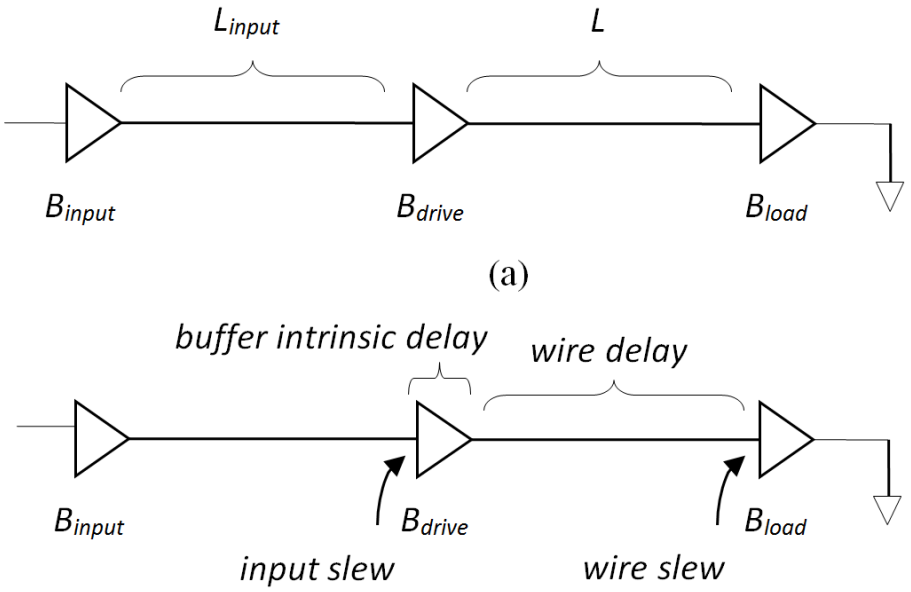

Figure 3.3: (a) Single-wire circuit structure used for simulation. (b) Delay and slew to measure.

For each combination of driving buffer type and load buffer type, we swept a range of different lengths for L_{input} and L and measured the input slew, buffer intrinsic delay, wire delay, and wire slew. Figure 3.3(b) shows the data we measured. Input slew and wire lengths are continuously distributed in clock trees, so we needed to build a continuous delay library based on the data we collected from SPICE simulations. We used MATLAB to perform surface-fitting (3-D curve fitting) on the collected data, that is, buffer intrinsic

delay, wire delay, and wire slew, with respect to input slew and length, and obtained a set of delay estimation functions. These functions are 3rd- or 4th-order polynomials in terms of input slew and length, which are sufficient for the range of input slew and lengths we need. Figure 3.4 is an example how we did surface fitting to obtain buffer intrinsic delay for one combination of driving buffer type and load buffer type based on different input slew and load wire lengths. Similar surface fitting is carried out for each combination for driving and load buffers.

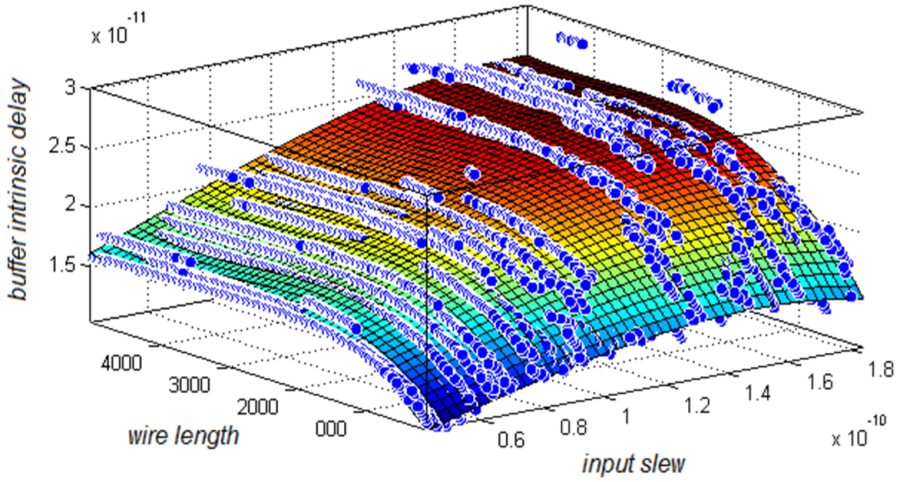

Figure 3.4: Buffer intrinsic delay as a function of input slew and wire length.

3.2.2 Modeling Branch-Type Components

For branch-type components, we consider only cases with two branches for the sake of simplicity. An example circuit with two branches of different lengths is shown in Figure 3.5. The clock trees generated by our CTS algorithm are always binary, so it is sufficient to model only cases with two branches. The data of wire delays and wire slews of both branches are collected. We performed hyperplane-fitting instead of surface-fitting on SPICE simulation data due to the increase of dimensions. Figures 3.6 and 3.7 are examples of the hyperplane fitting in three of the possible dimensions. The resulting polynomial functions are used similarly as in the case of single-wire-type components.

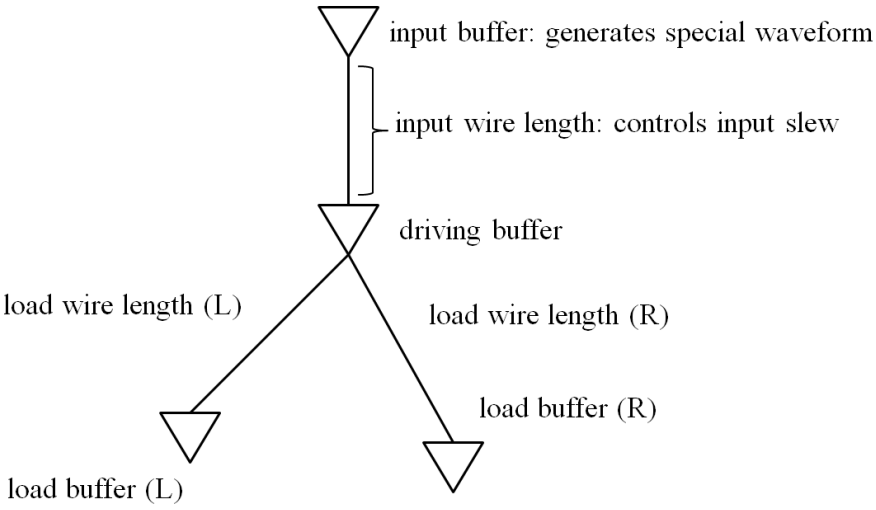

Figure 3.5: Branch circuit structure used for simulation.

3.2.3 Use of Delay/Slew Library

Whenever there is a need to compute delay or slew on a single-wire-type or a branched-type component, the set of functions corresponding to the specified driving and load buffer types can be used to compute highly accurate delay and slew values that are comparable to SPICE simulation results.

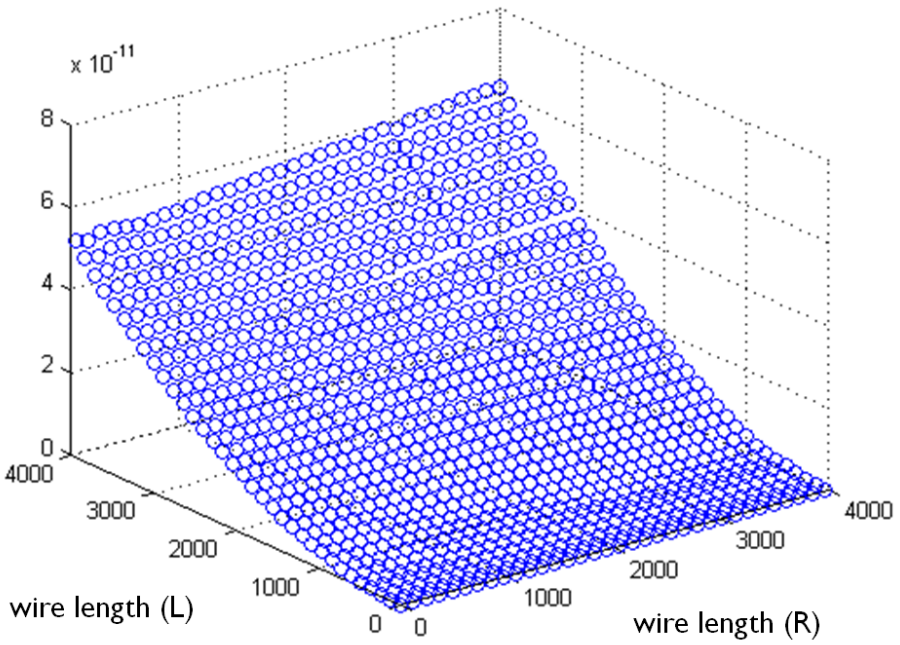

Figure 3.6: Wire delays of the left branch as a function of left and right wire lengths.

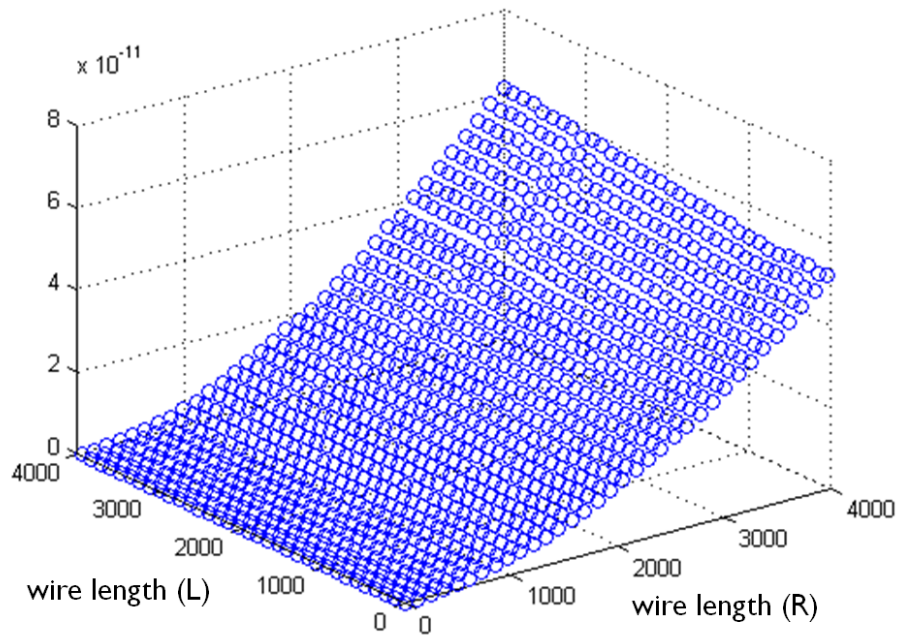

Figure 3.7: Wire delays of the right branch as a function of left and right wire lengths.

CHAPTER 4

BUFFERED CLOCK TREE SYNTHESIS ALGORITHM

The problem we are to solve by our proposed algorithm can be formulated as follows: Given a set of clock sinks, different types of buffers, and a single type of wire, we want to synthesize a buffered clock tree such that the slew in any part of the clock tree is bounded, and the clock skew is minimized.

4.1 Top-Level Algorithm

4.1.1 Levelized Topology Generation

Our clock tree synthesis framework is a levelized adaptation from [3]. The levels are defined as follows: The sinks are in level 0, the merge nodes obtained from merging the sinks are in level 1, and so on. The root of the clock tree is at the highest level. Building a levelized clock tree helps reduce the skew because the tree itself is more balanced. Figure 4.1 illustrates our top-level algorithm.

Initially, a nearest-neighbor graph is constructed based on a cost function that is able to capture the feasibility of pairing the nodes and merging them into a sub-tree. In order to consider all combinations, the nearest-neighbor graph is built as a complete graph. The edge cost is a function of distance and delay difference between each pair of nodes. If two nodes are close to each other, only a short segment of wire is needed to connect them. As a consequence, it is helpful to choose node pairs with smaller distance so as to reduce the total wire length, which further reduces delay and power in the final clock tree. Small delay difference is also desired because it helps balance delays on the two sides when generating the merge node. Therefore, we have

```
Given a set of clock sinks S
BufferedClockTreeSynthesis(S)
{
   V ← S;
   G ← InitializeNearestNeighborGraph(V);
   while (|V| > 1) {
      E ← FindMatching(G);
      for each e ∈ E {
         (v1, v2) ← e.endpoints;
         M ← MergeRouting(v1, v2);
         G ← UpdateNearestNeighbors(v1, v2, M);
      }
   }
   return V; // contains only the root node of the
clock tree
}
```

Figure 4.1: Top-level algorithm.

the edge cost of an edge e connecting nodes v_1 and v_2 as follows:

$$cost(e) = \alpha \cdot distance(v_1, v_2) + \beta \cdot |delay(v_1) - delay(v_2)| \qquad (4.1)$$

After the nearest-neighbor graph is constructed, we find a matching (a set of edges with no shared endpoints) of minimum total cost in the graph. The matching represents the pairs of sub-trees that are most feasible for merging at the current level. We process (merge) each of the edges in the matching and obtain merge nodes as new candidates for pairing. The processed edges are removed from the nearest-neighbor graph, and the new merge nodes are added into the graph. When all edges in the matching are processed, we complete building a level. Then, we find another matching in the updated nearest-neighbor graph and continue to build the next level. The algorithm terminates when only one node, which is the root of the clock tree, is left in the nearest-neighbor graph, indicating that all sinks are embedding into a single tree.

Our algorithm to find a matching is a greedy heuristic. First, we compute the centroid of all sink coordinates. Then, we repeatedly choose the node which is farthest from the centroid and its nearest neighbor to form a pair. In the case of having odd numbers of nodes, a seed node is chosen not to be matched with other nodes and is directly transferred to the next level. The

seed node is chosen as the node with maximum latency due to the fact that the nodes in the next level have larger delays. In this way, delays can be more balanced when the seed node and another node are merged in the next level. This approach outperforms the greedy algorithm introduced in [22].

Unlike [3] and other previous work, merging is performed using a proposed algorithm, merge-routing, rather than using the merge segment calculation method discussed in Section 2.2. The merge-routing algorithm finds a path from the root of each sub-tree to an estimated merge node and inserts buffers along the path in order to maintain reasonable slew. The details of merge-routing are discussed in Section 4.2.

4.1.2 H-Structure Correction

As seen in Figure 2.2, the order in which the same set of nodes are embedded greatly affects the topology, structure, and performance of the resulting clock tree. Unfortunately, it is difficult to predict the resulting topology as the CTS process is bottom-up. Nevertheless, it is possible to find an intertwined sub-tree in the CTS process after a few merging iterations take place. We added a checking feature in our CTS flow in order to remedy this situation in the case that undesirable pairings occur.

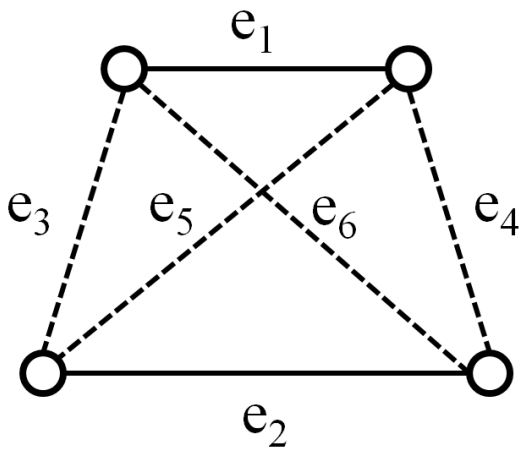

Figure 4.2: Three possible pairings of four nodes: (e_1, e_2), (e_3, e_4), and (e_5, e_6).

We check a pair of sub-tree's four children as illustrated in Figure 4.2 on how the children are paired.

Method 1 (Re-Estimation):

- Estimate the costs of the edges, e_1–e_6.
- Choose the pair of edges with lowest cost (delay difference).
- Perform merge-routing on the pair.

Method 2 (Correction):

- Perform merge-routing on all edges, resulting in 6 merge nodes, n_1–n_6.
- Choose the pair of merge nodes with lowest skew:
  $arg\min(\max(skew(n_1), skew(n_2)),$
  $\max(skew(n_3), skew(n_4)), \max(skew(n_5), skew(n_6)))$
- $\max(skew(n_1), skew(n_2))$: potentially, the skew of the merge node of n_1 and n_2 depends on $\max(skew(n_1), skew(n_2))$.
- Delete all other pairs.

Experimental results of adopting Methods 1 and 2 on top of the original algorithm are reported in Section 5.2.

4.2 Merge-Routing Algorithm

The proposed merge-routing algorithm consists of three stages: balance, route, and binary search. The balance stage, which is a preprocessing stage, reduces the difference of delays before two sub-trees are routed. The routing stage finds two buffered routing paths that have minimum skew. The binary search stage adjusts the location of the merge node so that the skew is reduced to the minimum.

4.2.1 Balance Stage

When the top-level algorithm selects a pair of sub-trees to merge, there is a limited amount of delay that merge-routing can balance without taking detours, as discussed in Section 2.2. The amount can be roughly estimated based on the distance of the roots of the sub-trees. Even though the proposed merge-routing algorithm deals not only with wires but also buffers, the slew constraint can be utilized to estimate the number of buffers that need to be inserted along the routing path, and thus we can have a rough estimation of buffer delay and wire delay that can be achieved without taking detours during merge-routing. If this amount is not sufficient to balance the delay difference of the two sub-trees, wire-snaking needs to be performed.

In order to honor the slew constraint during wire-snaking, we propose a progressive approach that inserts wires and buffers alternatively until the target delay is achieved. First, a segment of wire starting with a driving buffer is inserted. The wire length is gradually increased but it stops before the slew at the end of the wire exceeds the slew limit or the target delay is reached. The process repeats until the target delay is finally achieved. The new starting buffer acts as the new root of the sub-tree.

4.2.2 Routing Stage

The routing algorithm adopts the approach of global routing in standard cell design. The routing area has been defined and partitioned into routing grids. In contrast to the standard cell global routing, routing congestion is not the top priority in clock tree synthesis. Therefore, during routing, it is not necessary to keep global congestion information. Another feature of clock tree routing is that all the nets are two-pin nets and the length of nets varies over a wide range.

Based on the above observations, several innovations have been made in our clock tree routing framework. First, to balance accuracy and running time of widely distributed net lengths, we dynamically adjust the routing grid size R based on distance of each pair of nodes to be routed. As shown in Figure 4.3, S_1 and S_2 are two nodes that need to be merged and the region

between them is partitioned into grids. By default, we set an initial grid size $R = 45$ per dimension of bounding box. However, if the distance of two merging nodes is large, the routing grid size can increase dynamically for the purpose of evaluating more potential buffer locations, which provides more precise slew control. Therefore, for any net length, we can guarantee enough buffer insertion locations as well as maintain a steady routing time.

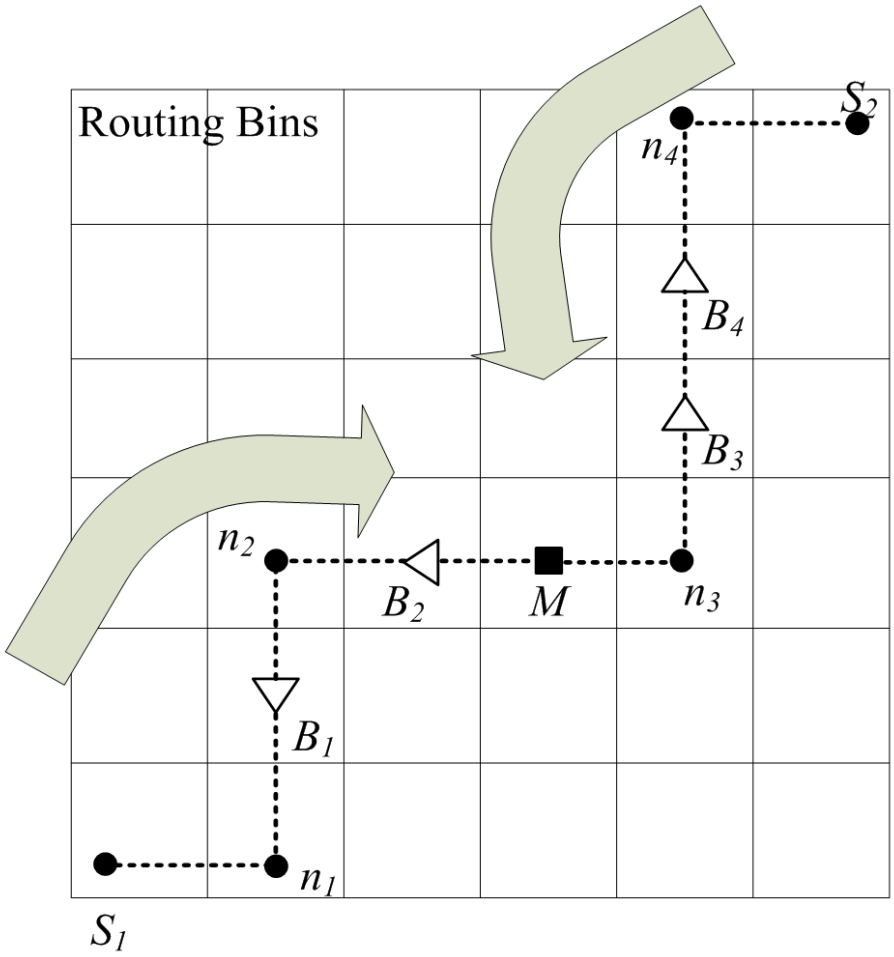

Figure 4.3: Bi-directional maze-routing. Routing path P_1 contains all buffered and unbuffered nodes from S_1 to M. Therefore, $P_1 = S_1, n_1, B_1, n_2, B_2$. Similarly, $P_2 = S_2, n_4, B_4, B_3, n_3$. The last fixed nodes of P_1 and P_2 are B_2 and n_3, respectively.

The second major innovation of this work is the bi-directional maze routing. Instead of routing from one sink to another, routing starts from two sinks (S_1 and S_2 as shown in Figure 4.3) simultaneously. By applying bi-

directional routing, each routing grid contains propagation delay information to both sinks. The grid with minimum delay difference (minimum skew) can be picked as a tentative merger location. In contrast to most existing clock tree synthesis algorithms which only insert buffer at merge node, in this work, buffers are inserted along wires to keep slew within the limit. To accurately compute propagation delay and slew, a pre-characterized delay library created by SPICE is used in this work for delay and slew look-up. This library is indexed by several design parameters as shown in Section 3.2.

The procedure of maze expansion and delay library lookup is demonstrated in Figure 4.4 as an example:

- Find load capacitance of wire segment L. In this example, load capacitance is the gate capacitance of Type 1 buffer.
- During routing, the length of wire segment L increases gradually.
- For each L, find its segment propagation delay from pre-computed delay estimation functions by assuming the driving buffer input slew to be equal to the slew limit. The slew at the other end of the wire (location S) has been monitored as well.
- If at a certain point L_2, slew S approaches the predefined limit, it means one buffer needs to be inserted to restore the slew of segment L.
- Ideally, the buffer should be inserted at the location which makes slew S identical to the slew limit. However wire length and slew at S increase discretely during maze routing. To minimize this error, an intelligent buffer insertion procedure has been applied in this work by evaluating multiple types of buffers at and ahead of the maze expansion grid in question. For example, at location L_2, a Type 1 buffer could be inserted to limit the slew at location S. Meanwhile, the router evaluates routing grids ahead of L_2 to find out if there exists a better location. As shown in the example, a larger Type 2 buffer can be inserted at location L_1 which makes slew S closest to the slew limit.

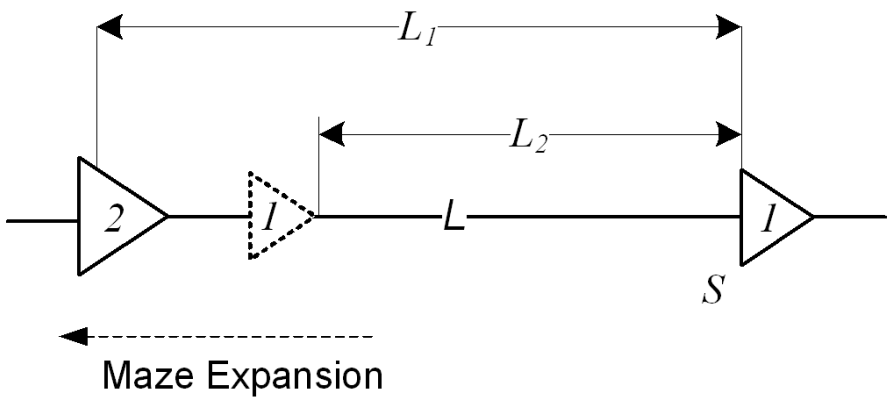

Figure 4.4: Maze expansion and intelligent buffer sizing.

4.2.3 Binary Search Stage

After the routing stage is completed, two arrays of routing path nodes, denoted as P_1 and P_2, are obtained (Figure 4.3). We perform binary search to find the optimum merge node M that minimizes the delay difference between P_1 and P_2. In the figure, B and n are routing nodes, where B means a buffer is inserted and n means no buffer is inserted. On path P_1, we first find the last fixed node $v_1 = B_2$ as the last node in P_1 that is not in the final routing bin. Similarly we can find $v_2 = n_3$ for path P_2. Let L be the line segment $v_1\bar{v}_2$. M is located on L based on a ratio r, which represents the ratio of the lengths of line segments $v_1\bar{M}$ and $v_2\bar{M}$. The ratio r is initially assigned to be 0.5. M is then moved along L according to top-down timing analysis results. Figure 4.5 shows the algorithm.

Because our SPICE-based timing analysis is more accurate than the Elmore delay model and is able to handle buffer delays, this method outperforms the merge node calculation method introduced in Section 2.2 in terms of accuracy.

4.3 Complexity Analysis

Let l be the longer dimension of the entire chip. Let n be the number of clock sinks. The number of merge nodes, which are the internal nodes in the final clock tree topology, is $n - 1$. Therefore, nearest-neighbor selection and

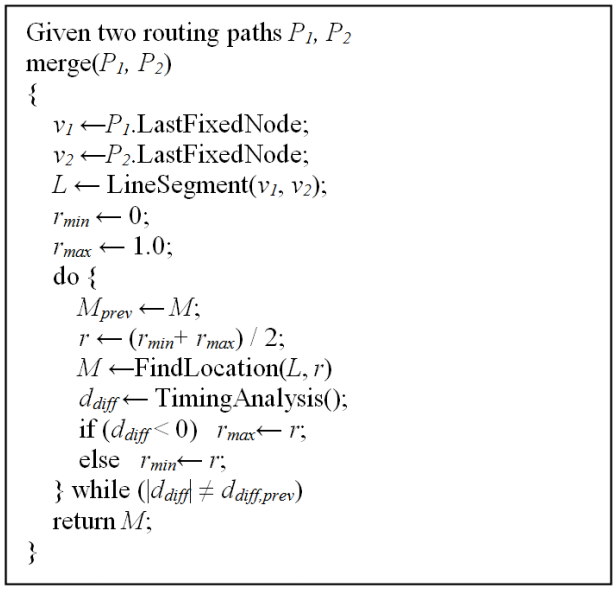

Figure 4.5: Binary search stage.

merge-routing are performed $n - 1$ times.

The time complexity for each nearest-neighbor selection is $O(n^2 \cdot \lg n)$ [22]. In the merge-routing algorithm, the balance stage is linear to the difference of delays, which can be considered as bounded. In the routing stage, the number of routing bins is bounded by $O(l^2)$, and the number of the resulting routing path nodes is bounded by the total number of routing bins. Therefore, the run time required for the routing stage is $O(l^2)$. The binary search stage is performed on the line segment with maximum length of $2l$. It terminates when the coordinate of M converges, which can be achieved in $O(\lg l)$ time.

It can be concluded from the above analysis that the running time bottleneck of our framework is the nearest-neighbor selection in topology generation. Overall, our algorithm requires $O(n^3 \cdot \lg n \cdot l^2)$ time if the clock tree topology is dynamically generated during the synthesis. However, if a fixed topology is given, our algorithm requires only $O(n \cdot l^2)$ time.

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 Experiments on Buffered Clock Tree Synthesis

We implemented our framework in C++. We used 45 nm PTM model as technology parameters. We performed experiments on benchmarks from the GSRC Bookshelf [23]. We used a library of 3 buffers, which are defined in transistor level using SPICE. Unit capacitance and unit resistance are 0.03 Ω/m and 0.2 fF/m, which are $10X$ bigger than what are specified in the benchmarks of [23]. This mimics bigger chips that incur stringent slew constraints. In this way, delay and especially slew grow much faster with the increase of wire length, and the need to insert buffers along routing paths is emphasized. Therefore, it is more suitable to test our algorithm in this setting. The experimental results are shown in Table 5.1. The slew limit is set as 100 ps, and we set it to 80 ps during synthesis in order to leave a margin. The worst slew, the skew, and the maximum latency are obtained from SPICE simulation of the clock tree netlist. The worst slew is the maximum slew among all nodes in the clock tree reported by SPICE. For every benchmark, it does not exceed the slew limit of 100 ps.

It is the best to compare our results with [7], since it also inserts buffers in the clock tree routing paths. However, the benchmarks used in [7] are not available to us. Some other works that integrate buffer insertion at the merge points in clock tree synthesis are [6, 8, 16], which are also suitable for comparison. Despite the fact that we have a much more difficult task of skew reduction and slew control by using $10X$ unit resistance and capacitance, our skew results are comparable to those in [6, 8, 16]. The skews of [6, 8, 16] are extracted from the experimental results reported in [16]. We also performed experiments on the benchmarks from the ISPD2009 Contest. These bench-

Table 5.1: Experimental results of GSRC benchmarks [23]. The numbers are in picoseconds.

		Our Results			Comparison		
	# of Sinks	Worst Slew	Skew	Max Latency	Skew [6]	Skew [8]	Skew [16]
r1	267	89.5	69.7	1.30	100	57.0	37.0
r2	598	89.3	59.9	1.69	96	87.4	59.5
r3	862	89.7	64.2	1.95	101	59.6	49.5
r4	1903	100.0	107.1	2.75	176	98.6	59.8
r5	3101	98.3	89.4	3.00	110	86.9	50.6

Table 5.2: Experimental results of ISPD benchmarks [24]. The numbers are in picoseconds.

		Our Results		
	# of Sinks	Worst Slew	Skew	Max Latency
f11	121	99.2	45.2	2.26
f12	117	83.6	45.8	1.92
f21	117	99.2	51.1	2.16
f22	91	100.0	42.4	1.62
f31	273	98.1	65.1	4.22
f32	190	85.2	52.3	3.38
fnb1	330	80.0	68.6	4.67

marks have large areas and it is very challenging to control slew. Table 5.2 shows the results. It can be noted that all skews are within 3% of maximum latency.

5.2 Experiments on H-structure Corrections

We added the H-structure correction features introduced in Section 4.1.2 on top of our original CTS flow. In the process, the cost functions of topology generation were adjusted to explore possible outcomes of H-structure re-estimation and corrections. Table 5.3 shows a typical outcome of adopting H-structure re-estimation and correction, respectively. The ratios are the skews coming from the algorithm in question with respect to those from the original algorithm. As larger skew means lower operating frequency, a positive, higher ratio indicates worsening of the quality of the clock tree, while a negative, lower ratio indicates improvement. The average ratio of re-estimation among

all 12 cases is -2.43%, while the average ratio of correction is -6.13%. The last column, # of flippings, shows how many pairs were actually corrected during synthesis. It can be observed that H-structure correction gives the best clock trees. Nevertheless, it is also the most computationally expensive, as all combinations need to be actually routed rather than simply evaluated by cost functions.

Table 5.3: Experimental results on H-structure corrections.

	Original	Re-Estimation		Correction		# of
	Skew (s)	Skew (s)	Ratio	Skew (s)	Ratio	Flippings
r1	5.31E-11	6.54E-11	23.07%	6.54E-11	18.75%	51
r2	5.26E-11	5.51E-11	4.79%	5.51E-11	4.57%	116
r3	7.44E-11	7.84E-11	5.32%	7.84E-11	5.05%	164
r4	8.66E-11	7.62E-11	-12.11%	7.62E-11	-13.78%	293
r5	7.95E-11	7.65E-11	-3.80%	7.65E-11	-3.95%	509
ispdf11	5.78E-11	4.52E-11	-21.68%	4.52E-11	-27.67%	19
ispdf12	3.71E-11	4.47E-11	20.69%	4.47E-11	17.14%	21
ispdf21	3.18E-11	4.00E-11	25.78%	4.00E-11	20.50%	22
ispdf22	4.31E-11	2.90E-11	-32.66%	2.90E-11	-48.50%	17
ispdf31	6.21E-11	5.63E-11	-9.32%	5.63E-11	-10.28%	44
ispdf32	6.37E-11	5.08E-11	-20.30%	5.08E-11	-25.47%	42
ispdfnb1	4.17E-11	3.79E-11	-8.99%	3.79E-11	-9.88%	71

CHAPTER 6

CONCLUSION

We have proposed a buffered clock tree synthesis algorithm. The potential buffer locations are not limited to merge nodes, and thus we are able to have robust slew control. Furthermore, we have overcome the difficulty and inaccuracy of delay and slew in a bottom-up buffer insertion scheme. The skew of our clock tree remains reasonable under such aggressive buffer insertion. Part of this work is published in [18].

REFERENCES

[1] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, and A. B. Kahng, "Zero skew clock routing with minimum wirelength," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 39, no. 11, pp. 799–814, Nov 1992.

[2] R. S. Tsay, "An exact zero-skew clock routing algorithm," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 12, no. 2, pp. 242–249, Feb 1993.

[3] M. Edahiro, "A clustering-based optimization algorithm in zero-skew routings," *Design Automation, 1993. 30th Conference on*, pp. 612–616, 1993.

[4] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. Tsao, "Bounded-skew clock and Steiner routing," *ACM Trans. Design Automation Electronic Systems*, vol. 3, no. 3, pp. 341–388, 1998.

[5] J. Chung and C.-K. Cheng, "Optimal buffered clock tree synthesis," in *ASIC Conference and Exhibit, 1994. Proceedings., Seventh Annual IEEE International*, Sep 1994, pp. 130 –133.

[6] Y. Chen and D. Wong, "An algorithm for zero-skew clock tree routing with buffer insertion," in *European Design and Test Conference, 1996. ED TC 96. Proceedings*, Mar 1996, pp. 230 –236.

[7] G. E. Tellez and M. Sarrafzadeh, "Minimal buffer insertion in clock trees with skew and slew rate constraints," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 16, no. 4, pp. 333–342, 1997.

[8] R. Chaturvedi and J. Hu, "Buffered clock tree for high quality IC design," in *Quality Electronic Design, 2004. Proceedings. 5th International Symposium on*, 2004, pp. 381 – 386.

[9] G. Venkataraman, N. Jayakumar, J. Hu, P. Li, S. Khatri, A. Rajaram, P. McGuinness, and C. Alpert, "Practical techniques to reduce skew and its variations in buffered clock networks," *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*, pp. 592–596, 2005.

[10] A. Vittal and M. Marek-Sadowska, "Low-power buffered clock tree design," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 16, no. 9, pp. 965–975, Sep 1997.

[11] I.-M. Liu, T.-L. Chou, A. Aziz, and D. F. Wong, "Zero-skew clock tree construction by simultaneous routing, wire sizing and buffer insertion," in *Proceedings of the 2000 International Symposium on Physical Design*, ser. ISPD '00. New York, NY, USA: ACM, 2000, pp. 33–38.

[12] J.-L. Tsai, T.-H. Chen, and C.-P. Chen, "Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, no. 4, pp. 565 – 572, April 2004.

[13] L. Bing, H. Jiang, E. Gary, and S. Haihua, "Process variation aware clock tree routing," in *ISPD '03: Proceedings of the 2003 International Symposium on Physical Design*. New York, NY, USA: ACM, 2003, pp. 174–181.

[14] L. W.-C. Douglas and K. Cheng-Kok, "Process variation robust clock tree routing," in *ASP-DAC '05: Proceedings of the 2005 Conference on Asia South Pacific Design Automation*. New York, NY, USA: ACM, 2005, pp. 606–611.

[15] U. Padmanabhan, J. M. Wang, and J. Hu, "Robust clock tree routing in the presence of process variations," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 8, pp. 1385–1397, 2008.

[16] R. Anand and P. D. Z., "Variation tolerant buffered clock network synthesis with cross links," in *ISPD '06: Proceedings of the 2006 International Symposium on Physical Design*. New York, NY, USA: ACM, 2006, pp. 157–164.

[17] A. Rajaram and D. Z. Pan, "Fast incremental link insertion in clock networks for skew variability reduction," *Quality Electronic Design, 2006. ISQED '06. 7th International Symposium on*, pp. 6 pp.–84, 2006.

[18] Y.-Y. Chen, C. Dong, and D. Chen, "Clock tree synthesis under aggressive buffer insertion," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, June 2010, pp. 86 –89.

[19] X.-W. Shih and Y.-W. Chang, "Fast timing-model independent buffered clock-tree synthesis," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, June 2010, pp. 80 –85.

[20] C. Alpert, F. Liu, C. Kashyap, and A. Devgan, "Closed-form delay and slew metrics made easy," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, no. 12, pp. 1661 – 1669, Dec 2004.

[21] C. V. Kashyap, C. J. Alpert, F. Liu, and A. Devgan, "Peri: a technique for extending delay and slew metrics to ramp inputs," in *Proceedings of the 8th ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, ser. TAU '02. New York, NY, USA: ACM, 2002, pp. 57–62.

[22] D. E. Drake and S. Hougardy, "A simple approximation algorithm for the weighted matching problem," *Inf. Process. Lett.*, vol. 85, no. 4, pp. 211–213, Feb 2003.

[23] A. B. Kahng and C.-W. A. Tsao, "VLSI CAD software bookshelf: Bounded-skew clock tree routing," 2000. [Online]. Available: http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/BST/

[24] C. Sze, "ISPD 2009 clock network synthesis contest," 2009. [Online]. Available: http://ispd.cc/contests/09/ispd09cts.html